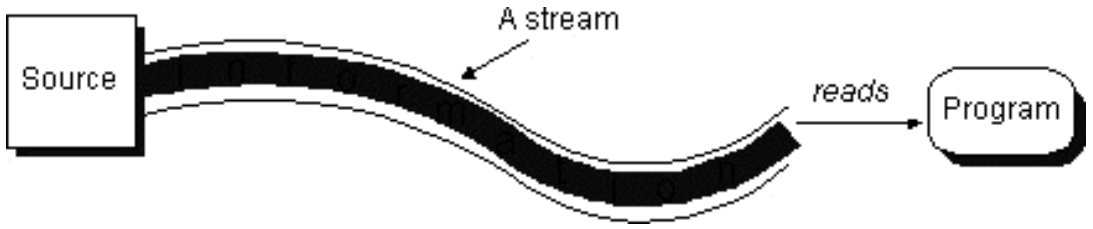
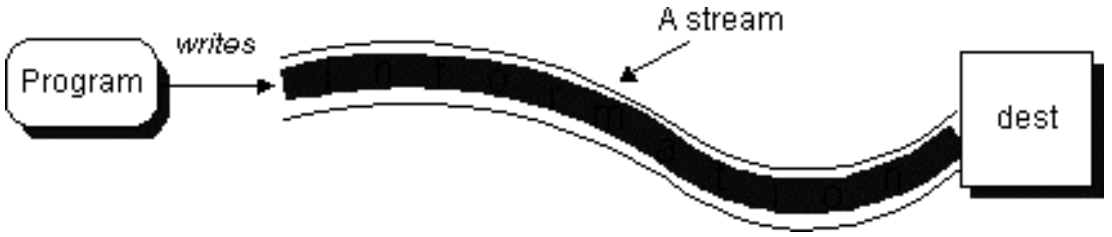
Ficheros

- Se utilizan para almacenar datos de forma permanente y que estos persistan después de la finalización del programa.
- La clase *File* y otras del paquete java.io permiten gestionar el acceso a ficheros.
- La E/S se define en términos de streams o flujos de datos que tienen una fuente (input streams) o un destino (output streams)

Input stream



Output stream



Clase File: Constructores

- La clase File se suele utilizar para manejar el fichero como un todo sin utilizar los datos que contiene.
- Constructores
  - Para dar el nombre del fichero o directorio:

```
File f1 = new File("/");
```

  - Para dar por una parte el fichero y por otra el directorio

```
File f2 = new File("/”, “autoexec.bat”);
```

  - Dar el nombre y una referencia al directorio

```
File f3 = new File(f1, "autoexec.bat");
```

Clase File: Métodos

- **String getName():** Nombre del fichero o del directorio
- **String getPath():** Nombre incluido el path del fichero
- **String getAbsolutePath():** Nombre del fichero incluyendo la dirección
- **String getParent():** Nombre del directorio padre del fichero
- **boolean exists()** true si el directorio o fichero existe
- **boolean canWrite()** true si el fichero se le puede escribir
- **boolean canRead()** true si el fichero se puede leer
- **boolean isDirectory()** true si el fichero es un directorio
- **boolean isFile()** true si el objeto es un fichero
- **boolean isAbsolute()** true si al constructor se le pasó dir. absoluta
- **long length()** Longitud del archivo en bytes
- **String[] list()** Array con el nombres de los ficheros del directorio
- **void delete()** Borra el fichero (Sólo para ficheros)
- **boolean renameTo(File destino)** true si se ejecuta con éxito (para direct o fichs.). También para moverlos si el fichero destino no existe.
- **boolean createNewFile()** Crea un nuevo fichero (sólo en versiones>=1.2)
- **boolean mkdir()** Crea un nuevo directorio
-

Clase File: Ejemplo

```
File f1, f2, f3,f4;
String[] listaFicheros;


int i;
f1= new File("C:\\", "autoexec.bat"); //doble barra:\ es especial
f2= new File(”pepe.xxx"); //no lo crea físicamente en disco
f3= new File(“dir"); //directorio actual
f4= new File(“fich_nuevo”);
System.out.println(f1.getName());       //Imprime "autoexec.bat"
System.out.println(f1.getPath());       //Imprime "C:\autoexec.bat
System.out.println(f2.getAbsolutePath());//Imprime”C:\java\pepe.xxx"
System.out.println(f1.getParent());     //Imprime "C:\"
System.out.println(f1.exists());        //Imprime true
System.out.println(f1.canWrite());      //Imprime true
f4.createNewFile();//Crea fich_nuevo sino existía, sino da error
System.out.println(f1.isDirectory());   //Imprime false
System.out.println(f1.isFile());        //Imprime true
System.out.println(f1.isAbsolute());    //Imprime true
System.out.println(f1.length());        //Imprime 429
listaFicheros=f3.list(); // si f3 no es un directorio da ERROR
```

```
for(i=0;i<listaFicheros.length;i++)
   System.out.println(listaFicheros[i]);//Imprime un fich. o directorio
```

Lectura/Escritura

- Independientemente del tipo de fichero los algoritmos para leer o escribir suelen tener el formato siguiente:

Lectura

Abrir el fichero

Mientras existan datos

    Leer datos

    Procesar datos

Cerrar el fichero

Escritura

Abrir el fichero

Mientras existan datos

    Escribir datos

Cerrar el fichero

- Existen varios tipos de streams y para cada casi todos los tipos de stream de lectura existe su correspondiente stream de escritura y viceversa por lo que generalmente se usan por

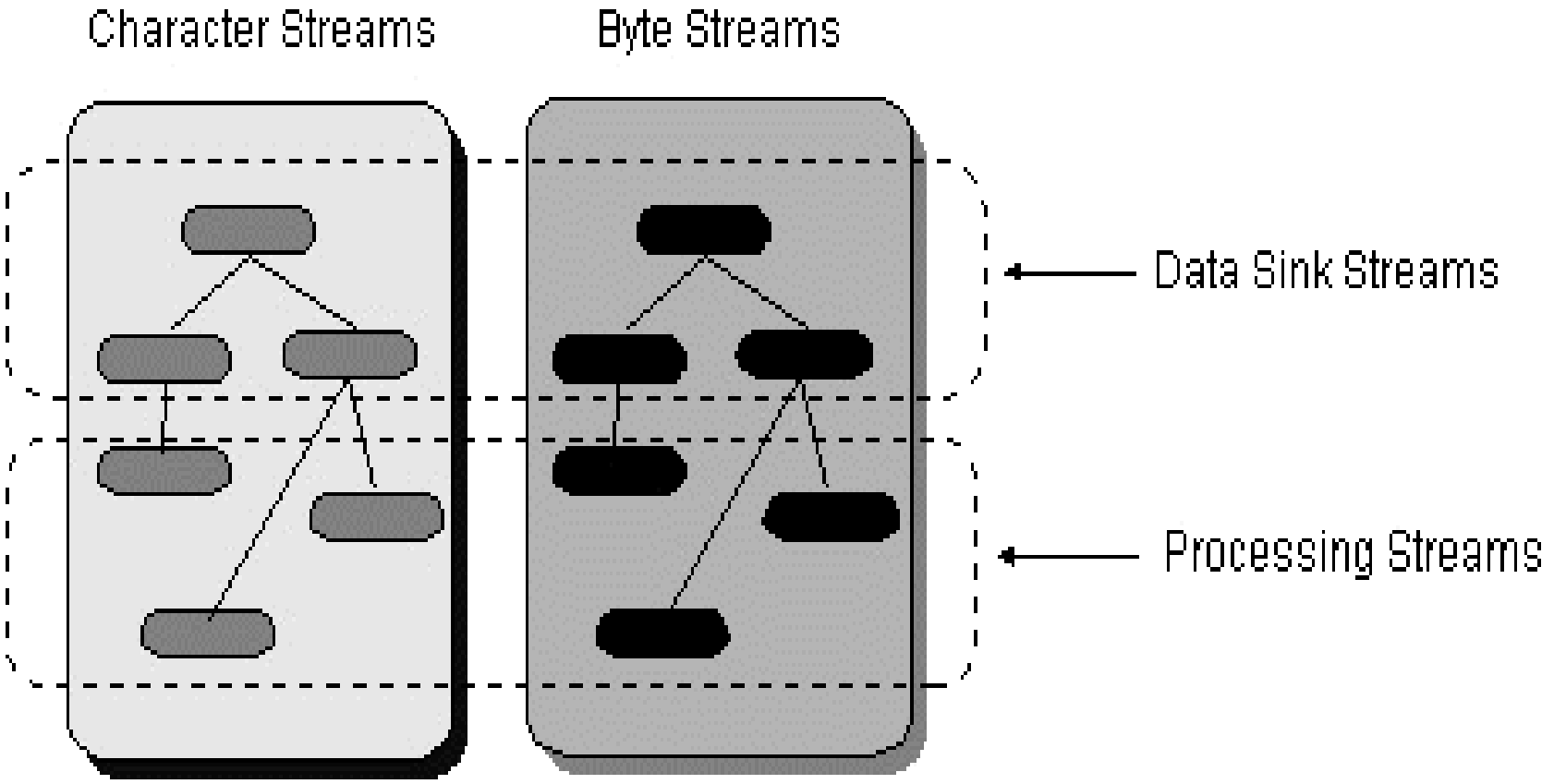
parejas

Lectura/Escritura

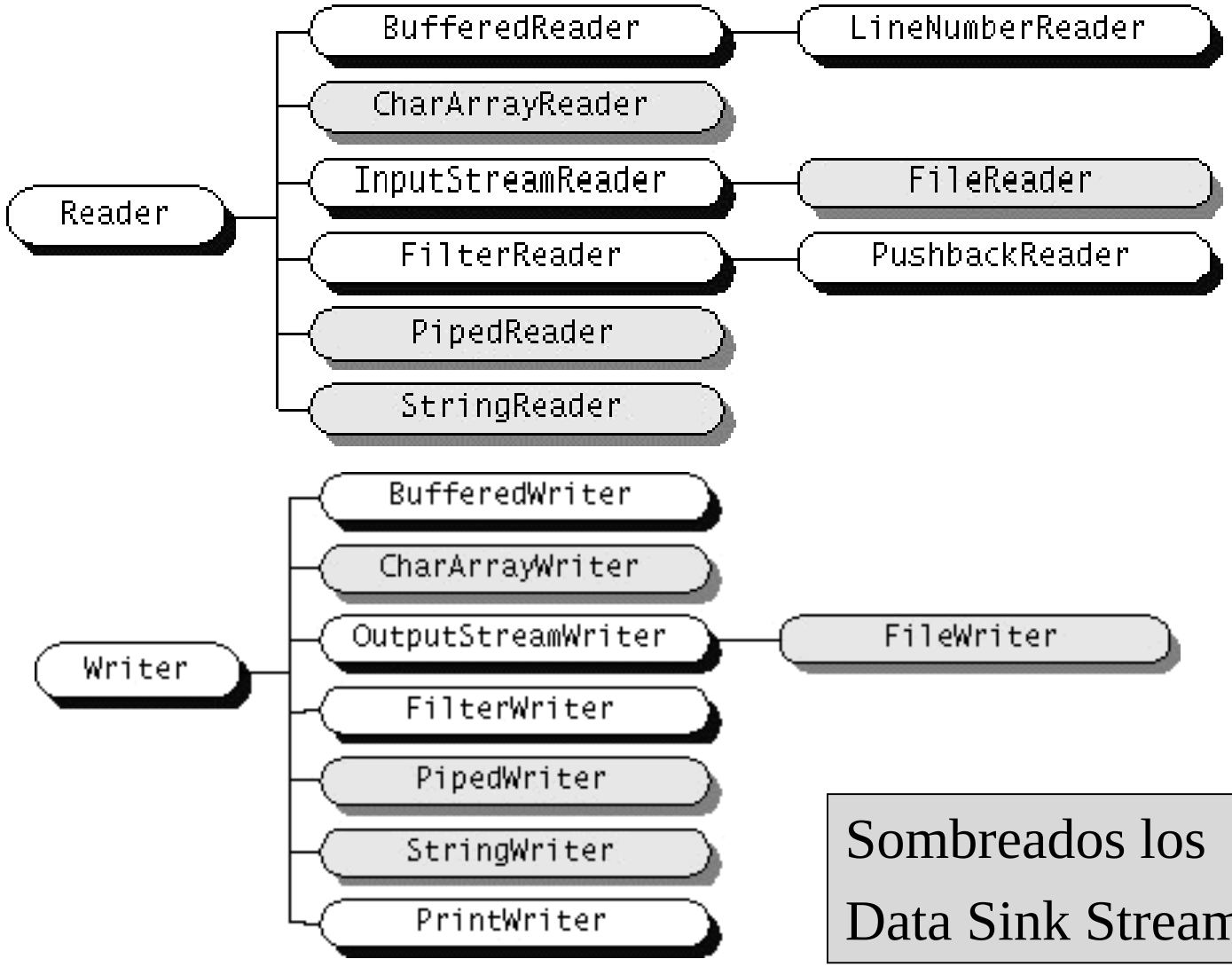
Tipos de Streams I

- El paquete java.io tiene 2 jerarquías de clases que dividen los streams según su contenido en:
  - **Character Streams** (16 bits, UNICODE) Utilizados para ficheros de texto, teclado, etc. Se les llaman readers o writers
  - **Byte Streams** (8 bit) Utilizados para ficheros binarios (imágenes, sonidos, …). Se les llaman input streams o output streams. System.in, System.out y System.err son byte Streams (se crearon antes de que se inventaran los Character Streams)
- También se puede considerar la división por el propósito (a título informativo).
  - **Data Sink Streams** : son fuentes u orígenes de datos
  - **Processing Streams** : Realizan algún tipo de operación de procesamiento (almacenarlos en buffer, descodificar caracteres,…) de los datos leídos o escritos de una fuente (Necesitan estar asociados a un Data Sink Stream)

Tipos de Streams II

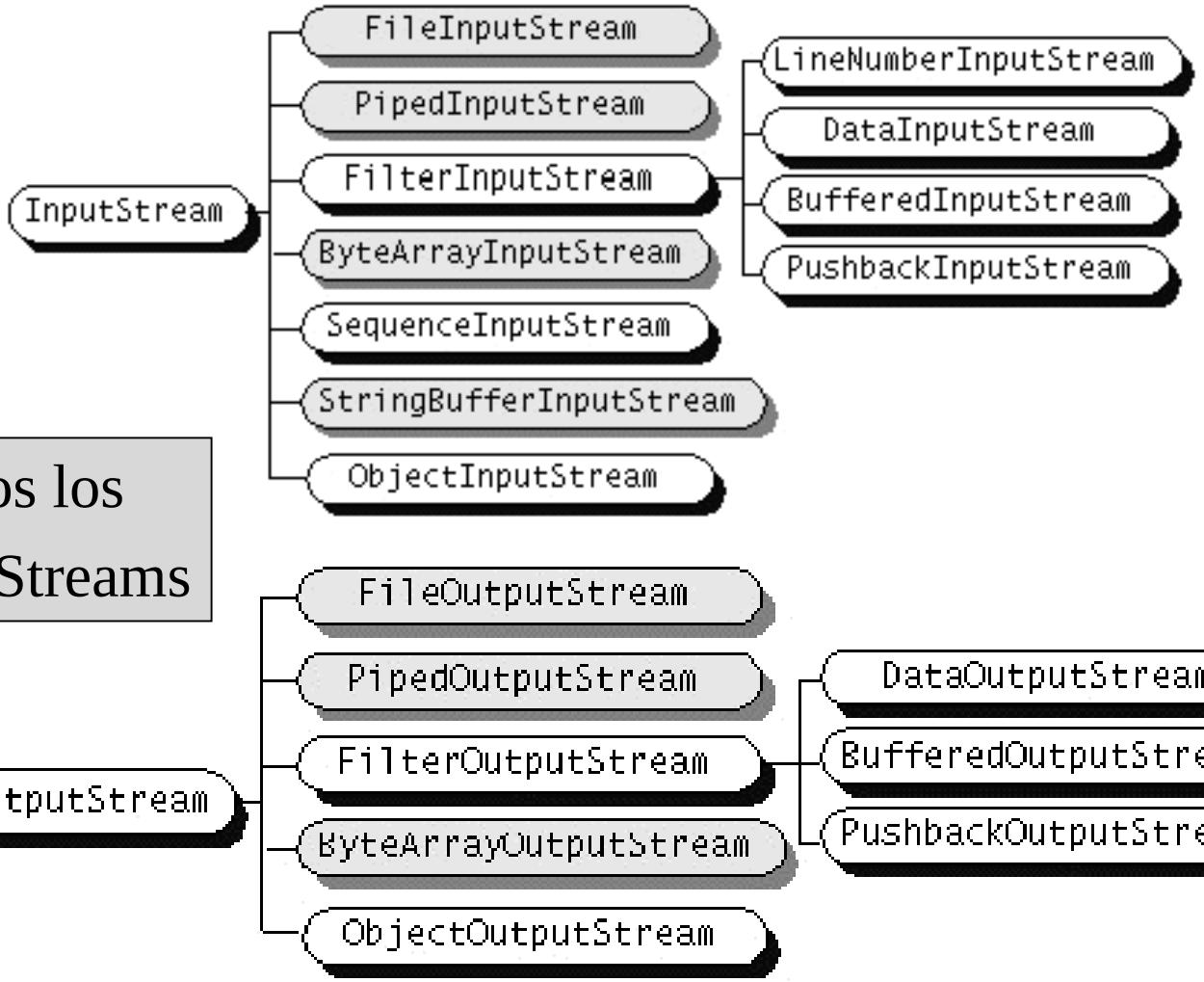


Character Streams



Sombreados los Data Sink Streams

Byte Streams



Sombreados los Data Sink Streams

FileInputStream/FileOutputStream

- Crean flujos de entrada/salida asociados a archivos de los que se leerán o escribirán bytes
- Son fuente o destino de datos (**Data Sink Streams**).
- Son subclases de las clases abstractas InputStream y OutputStream (las cuales lanzan IOException si se produce alguna condición de error).
- **Constructores**
  - `FileInputStream(String nombrefichero)`
  - `FileInputStream(File un_objeto_file)`
  - `FileOutputStream(String nombrefichero)`
  - `FileOutputStream(File un_objeto_file)`
  - `FileOutputStream (String nombrefichero, boolean añadir)`: Si añadir es true lo escrito se añade al final del fich.
- Si al crear un flujo de salida el archivo de destino en cuestión ya existe, se pierden los datos que contuviera, si no existe se crea un fichero nuevo.
- Si al crear un flujo de entrada el archivo no existe se genera FileNotFoundException.

Métodos de FileInputStream

- **int read()** Lee un byte y devuelve su valor convertido a entero. Si se llega al fin del stream devuelve -1
- **int read(byte b[])** Intenta leer *b.length* bytes y devuelve el número de bytes leídos, almacenando en b lo leído.
- **int read(byte b[], int offset, int len)** Lee hasta *len* bytes y los coloca en *b* a partir de la posición *offset*.
- **int skip(long n)** Salta n bytes de la entrada, devuelve el número de bytes omitidos
- **int available()** Devuelve el número de bytes actualmente disponibles en la entrada. Se suele emplear para comprobar que aún existen datos en el fichero antes de proceder a la lectura. De esta forma se evita tener que utilizar la excepción EOFException que marca el fin de fichero.
- **void close()** Cierra el stream (conviene invocarlo cuando el stream no vaya a usarse más)

Métodos de FileOutputStream

- **void write(int b)** Escribe un único byte
- **void write(byte b[])** Escribe un array de bytes
- **void write(byte b[],int offset, int len)** Escribe *len* bytes de *b* desde *b[offset]*
- **void flush()** Vacía el buffer de escritura del fichero, enviándo los datos del buffer al sistema operativo. Lo habitual es que se escriban en el disco.
- **void close()** Cierra el stream vaciando sus búfferes (conviene invocarlo cuando no vaya a usarse más)

- *Al igual que los métodos de FileInputStream, éstos lanzan IOException*

DataInputStream/DataOutputStream

- Se utilizan para realizar E/S utilizando directamente los tipos primitivos int, double, ...(sin convertirlos a bytes)
- Son Streams de procesamiento y por tanto necesitan estar asociados a un Data Sink Stream
- **Constructores** (reciben como parámetro un InputStream, un OutputStream o alguno de sus descendientes (ej. FileInputStream))

```
DataInputStream( InputStream un_objeto_InputStream)

DataOutputStream(OutputStream un_objeto_OutputStream)
```

- Ejemplo:

```
DataInputStream mi_flujo = new DataInputStream(new
  FileInputStream( "nombre.ext" ));

DataOutputStream otro = new DataOutputStream(new
  FileOutputStream( "nombre.ext" ));
```

Métodos de DataInputStream

Métodos que leen tipos primitivos:

boolean readBoolean()	**byte readByte()**
char readChar()	**double readDouble()**
float readFloat()	**int readInt()**
long readLong()	**short readShort()**

int read(byte b[], int off, int len): Lee hasta len bytes y los guarda en b desde la posición off. Devuelve cuantos ha leído

String readUTF(): Lee cadenas en Unicode, compactadas en UTF-8

int skipBytes(int n): Saltar n bytes

void readFully(byte array[]): Lee hasta llenar el array

void close()

UTF (8 bits)(Unicode Independent Code) Código independiente de la máquina que escribe por cada cadena, primero 2 bytes con la longitud en bytes de la cadena y después los caracteres de la misma (con 1, 2 o 3 bytes por carácter dependiendo del valor Unicode del carácter).

Métodos de DataOutputStream

Métodos que escriben tipos primitivos (devuelven void):

writeBoolean(boolean b)	**writeByte(byte b)**
writeChar(char b)	**writeDouble(double b)**
writeFloat(float b)	**writeInt(int b)**
writeLong(long b)	**writeShort(short b)**

void write(int b): Escribe el byte o array de bytes (int o compatible)

void writeChars(String s): Escribe s en el fichero

void writeBytes(String s): Escribe s en el fichero eliminando el byte superior de cada carácter de s

void writeUTF(String s) Escribe cadenas en Unicode, compactandolas en UTF(8 bits)

void flush (): Vacía el buffer

void close()

FileReader /FileWriter

- Son Character Streams similares a los Byte Streams InputStream y OutputStream
- Son fuente o destino de datos (**Data Sink Streams**).
- Se sincronizan por medio de un bloqueo para que si hay varios programas accediendo al mismo fichero no se produzcan inconsistencias
- **Constructores**
  - `FileReader(String nombrefichero)`
  - `FileReader(File un_objeto_file)`
  - `FileWriter (String nombrefichero)`
  - `FileWriter(File un_objeto_file)`
- Si al crear un nuevo flujo de salida el archivo de destino en cuestión ya existe, se pierden los datos que contuviera, si no existe se crea un fichero nuevo.
- Si se producen problemas al crear los flujos, se genera una excepción de tipo IOException.

Métodos de FileReader

- **int read()** Lee un carácter y devuelve su valor convertido a entero bloqueándose hasta que esté disponible, ocurra un error o se alcance el fin del stream. Si se llega al fin del stream devuelve -1
- **int read(char b[])** Intenta leer *b.length* caracteres y devuelve el número de caracteres leído o -1 si fin de fichero
- **int read(char b[], int offset, int len)** Lee hasta *len* caracteres y los coloca en *b* a partir de la posición *offset*.
- **int skip(long n)** Salta n caracteres de la entrada, devuelve el número de caracteres omitidos
- **boolean ready()** Devuelve true si el stream esta listo para ser leído, es decir si la siguiente invocación a read no se bloqueará
- **void close()** Cierra el stream

Métodos de FileWriter

- **void write(int b)** Escribe un único carácter
- **void write(char b[])** Escribe un array de caracteres
- **void write(char b[],int offset, int len)** Escribe *len* caracteres de *b* desde *b[offset]*
- **void write(String b)** Escribe un String
- **void write(String b,int offset, int len)** Escribe *len* caracteres del string *b* desde la posición *offset*
- **void flush()** Inicializa la salida garantizando que se vacíen todos sus búfferes
- **void close()** Cierra el stream

Al igual que los métodos de FileReader, éstos lanzan IOException

BufferedReader /BufferedWriter

- Se utilizan para evitar que cada lectura o escritura acceda directamente al fichero ya que utilizan un buffer intermedio entre la memoria y el stream
- Son Streams de procesamiento y por tanto necesitan estar asociados a un Data Sink Stream
- **Constructores** (reciben como parámetro un Reader, un Writer o alguno de sus descendientes (ej. FileReader o FileWriter)

```
BufferedReader(Reader unobjetoReader)
BufferedWriter(Writer unobjetoWriter)
```

- Ejemplos:

```
BufferedReader entrada = new BufferedReader( new
   FileReader("fin.dat") );
BufferedReader entrada2 = new BufferedReader( new
   FileReader(new File("fin2.dat")) );
BufferedWriter salida = new BufferedWriter( new
   FileWriter("fout.dat") );
BufferedWriter salida2 = new BufferedWriter( new FileWriter(new
   File("fout2.dat") ) );
```

Métodos de BufferedReader/BufferedWriter

- Los métodos básicos que proporcionan una funcionalidad añadida a estos streams son:
  - **String readLine()** de BufferedReader que lee una línea del Stream y la devuelve como resultado, si se llega al fin de fichero devuelve *null*.
  - **newLine()** de BufferedWriter que escribe un salto de línea en el stream

- Para otras operaciones se pueden usar los métodos de FileReader o FileWriter. Por ejemplo para escribir un número real (float)

```
salida.write((new Float(3.21)).toString() );
```

Ejem. Anadir datos al final de un BufferedWriter

```
import java.io.*;
/*Sistema para añadir al final de un BufferedWriter utilizando
   FileOutputStream*/
public class OutStrWri {
  public static void main(String arg[]) throws IOException {
  BufferedWriter bw=new BufferedWriter(new OutputStreamWriter(new
   FileOutputStream("OutStrWr.dat",true)));

    bw.write("Una linea\n"); /*no es exactamente igual que un
   newLine*/
    bw.write("Dos lineas");
    bw.newLine();
    bw.write("Tres lineas");
    bw.newLine();
    bw.flush();
    bw.close();
}
}
```

Lectura de Teclado

- Los Streams se pueden utilizar para Realizar lecturas y escrituras en la entrada y salidas standard aprovechando que son Character Streams.
- Para leer de teclado utilizando un Buffered Reader hay que filtrar la entrada estandar (System.in).
- Ejemplo:

```
InputStreamReader isr=new InputStreamReader(System.in);
BufferedReader br= new BufferedReader(isr);
String s2=br.readLine();
```

InputStreamReader y OutputStreamWriter transforman Byte streams en CharacterStreams

  - **InputStreamReader(InputStream in)** Recibe un byte stream como entrada y produce como salida los correspondientes caracteres unicode.
  - **OutputStreamWriter (OutputStream out)** Permite escribir los caracteres enviados en un byte stream.

Lectura de Teclado

- Otra forma de leer de teclado es con la clase Scanner:

```
static Scanner in=new Scanner (System.in);


static int leerInt(){
  int n;
  try {
    n=in.nextInt();
  } catch (NoSuchElementException ne) {
    return (0);
  }
  return (n);
}
```

Otros tipos de streams

- **LineNumberReader** Reader que facilita la numeración de las líneas leídas.
- **RandomAccessFile** que permite acceder posiciones del contenido del fichero de forma directa sin necesidad de realizar acceso secuencial. No hereda de InputStream ni OutputStream pero implementa los interfaces de DataInput y DataOutput
- **PrintStream**: streams de salida que facilitan la lectura de los datos escritos por parte del usuario. System.out es el único PrintStream que se puede utilizar y asume que todos los bytes escritos son caracteres Latin-1. Proporciona los métodos print y println
- **ObjectOutputStream** y **ObjectInputStream** para ficheros en los que se desean guardar objetos que implementan la interfaz Serializable.

ObjectOutputStream

- Escritura Entre try y catch (Exception e) pondremos:

```
ObjectOutputStream oo = new ObjectOutputStream(
        new FileOutputStream("ficheroDatos.dat"));
        for(int i=0;i<numProductos;i++){
          oo.writeObject(productos[i]); // donde Producto [] productos;
        }
        oo.close();
```

- La clase de los objetos a grabar debe tener implementado el interfaz Serializable e incluir un constructor sin parámetros
- Es conveniente incluir en las clases serializables el atributo **private static final long *serialVersionUID= 34;*** e ir modificándolo según cambie la versión.

```
public class Producto
implements Serializable {
   String nombre;
   double precio;

    Producto (){}
}
```

ObjectOutputStream

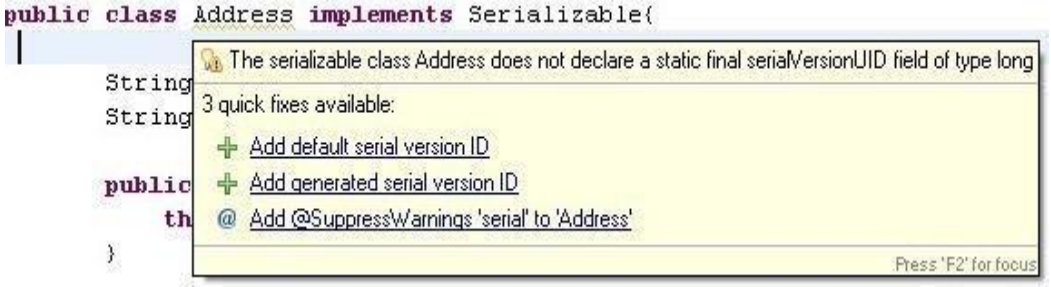
ObjectInputStream

```
int i=0;
ObjectInputStream oi=null;
try{
      oi= new ObjectInputStream(new FileInputStream("ficheroDatos.dat" ));
}catch (Exception e) {       System.out.println ("El fichero no existe");   }
Producto [] productos= new Producto[ 100];
try{
   boolean fin=false;
   while (!fin){
      try{
         productos[i]= (Producto) oi.readObject();
         i++;
      }catch (EOFException e) {          fin=true;       }
  }// fin del while
}catch (Exception e) {
   System.out.println ("Problemas con lectura del fichero");
}finally {   oi.close();   }
```

Interfaz Serializable

- La clase de los objetos a grabar con ObjectOutputStream debe tener implementado el interfaz Serializable e incluir un constructor sin parámetros. Si la clase tiene atributos de otras clases también estas deben ser serializables.
- Si cambian los atributos de la clase,en tipo o en número, entre la escritura y la lectura entonces no se podrá leer correctamente el fichero.
- Es conveniente incluir en las clases serializables el atributo **private static final long *serialVersionUID= 34;*** e ir modificándolo según cambie la versión o mejor, utilizar eclipse para que genere el número de forma automática colocando el cursor sobre la clase que implementa Serializable y elegir la generación del número de serie

Si se modifica la clase utilizada para grabar los objetos (modificando cualquier método) el SerialVersionUID cambiará y si no se ha definido fijo (con un atributo) no se podrán leer las clases grabadas en el fichero.



RandomAccessFile

- **RandomAccessFile** (sobre bytes) que permite acceder posiciones del contenido del fichero de forma directa sin necesidad de realizar acceso secuencial.
- No hereda de InputStream ni OutputStream pero implementa los interfaces de DataInput y DataOutput y por tanto **se pueden usar los métodos vistos para DataInputStream y DataOutputStream** que también lo implementan.
- Constructores
  - ***RandomAccessFile (File file, String mode)***
  - ***RandomAccessFile (String name, String mode)***
    Crea un fichero de acceso directo para leer y, opcionalmente, escribir en el.

RandomAccessFile

- Modos:
- "r" Lectura. Si se intenta escribir saltará IOException. Si el fichero no existe saltará FileNotFoundException
- "rw" Lectura y Escritura. Si el fichero no existe se creará.
- "rws" Abierto para lectura y escritura y toda escritura () actualizará datos y metadatos en el disco.
- "rwd" Abierto para lectura y escritura y toda escritura actualizará datos (no los metadatos)en el disco.

RandomAccessFile

- public void **seek**(long pos) throws IOException Coloca el puntero del fichero en el byte indicado por pos, contando desde el principio del fichero.
  - Se puede ir mas allá del final del fichero y si se escribe más allá del fin del fichero se cambiará el tamaño del fichero.

- public long **length**() throws IOException Devuelve el tamaño del fichero en bytes

- public long **getFilePointer**() throws IOException Devuelve la posición actual del puntero del fichero en bytes desde el principio del fichero